Attorney's Ref. No.: P214062

EVENT MANAGEMENT SYSTEMS AND METHODS FOR THE DISTRIBUTION OF MOTION CONTROL COMMANDS

RELATED APPLICATIONS

This application claims priority of U.S. Provisional Application Serial No. 60/267,645 filed February 9, 2001.

TECHNICAL FIELD

The present invention relates to motion control systems and, more specifically, to an event management system optimized for the processing and distribution of motion control commands for a motion control device.

BACKGROUND OF THE INVENTION

Electromechanical systems are used in numerous environments to translate electrical signals in mechanical movement. As examples, factory automation systems, toys, appliances, and the like all may use one or more electromechanical transducers that move in response to electrical control signals.

Typically, an application programmer familiar with a specific environment creates an application program defining a desired sequence of movements. U.S. Patent Nos. 5,691,897, 5,867,385, and 6,209,037 to Brown et al. disclose systems and methods for generating, processing, and/or distributing control commands to one or more motion control devices based on such an application program.

The present invention may be embodied as a part of an overall motion control system as described in the Brown et al. patents and will be described herein in that context. However, as described below, the principles of the present invention may have broader application to other motion control systems and methods, and the scope of the present invention should be determined by the claims appended hereto and not the following detailed description.

SUMMARY OF THE INVENTION

A motion control system for controlling a target device to perform a desired motion operation. The motion control system comprises at least one motion event provider, a motion event manager, and a motion control component. The motion event provider is configured to generate at least one event token upon the occurrence of at least one predetermined event. The event token is associated with at least one motion command. The motion event manager receives the at least one event token. The motion event manager uses the motion control component to transmit to the target device a control command based on the at least one motion command associated with the event token received by the motion event manager.

DESCRIPTION OF THE DRAWING

FIG. 1 is a module interaction map depicting a motion control system incorporating an event management system of the present invention;

FIG. 2 is a scenario map depicting the startup and initialization of the system of FIG. 1;

FIG. 3 is a scenario map depicting the process of injecting a motion event provider DLL into the address space of a target application;

FIG. 4 is a scenario map depicting the process of configuring motion events provided by each of one or more motion event providers 132;

FIG. 5 is a scenario map depicting the process of handling a motion event;

FIG. 6 is a scenario map depicting how a user may obtain new motion media and motion event providers 132 from the motion web site 136;

FIG. 7 is a scenario map depicting shut down of the system of FIG. 1; and

FIG. 8 is a scenario map depicting the mapping of text to motion events.

DETAILED DESCRIPTION OF THE INVENTION

Referring initially to FIG. 1, depicted therein is a motion control system 120 constructed in accordance with, and embodying, the principles of the present invention. The motion control system 120 is an event-based system used to control, configure, and query one or more motion based devices or machines such as indicated by reference character 122 in the drawing. The motion based devices or machines 122 will be referred to herein as the target device.

In the following discussion, the components or objects that form the motion control system 120 and the basic operation of the system 120 will first be described. After that will follow a discussion of the interaction between those objects and several detailed scenarios of typical actions of this system 120 .

Referring initially to FIG. 1 of the drawing, it can be seen that the motion control system 120 comprises the motion based machine or device 122, a motion event manager 130, one or more motion event provider objects 132, a motion event interface 134, a motion web site 136, and a media creation tool 138.

The system 120 is adapted to be run on a general purpose computer platform comprising computer hardware and an operating system 140. The exemplary operating system 140 is a Windows variant and comprises a registry 142.

The web site 136 and media creation tool 138 provide access to one or more motion media files 144. The motion media files 144 contain what will be referred to herein as motion media.

The term "motion media" includes motion scripts, motion application programs, and/or individual motion instructions capable of causing the target device 122 to perform a desired motion operation comprising a discrete movement or sequence of movements.

The motion media comprises what will be referred to as motion commands. The term "motion commands" will be used herein to refer to both control commands and media commands associated with a desired motion operation. The term "control commands" as used herein refers to device-specific commands that may be directly run by a target device to obtain a desired motion operation. The term "media commands" used herein refers to machine independent instructions that generically define a desired motion operation. Normally, media commands are converted into control commands before the target device executes the desired motion operation corresponding to a particular media command or set of media commands. The term "application program" will be used to refer to a set of control and/or media commands associated with a sequence of discrete movements.

The motion event manager 130 comprises a motion control component 150 and an event handling component 152. The motion event interface 134 further comprises a event provider configuration control 154 and a media view control 156.

The motion control system 120 operates basically as follows. The motion event providers 132 generate what will be referred to as event tokens based on the occurrence of a predetermined event. The event token is associated with the predetermined event in advance and thus identifies the predetermined event. The event token may also contain additional information such as the source of the predetermined event, parameters associated with the predetermined event, and the like.

The event tokens are sent to the motion event manager 130. The motion event providers 132 and motion event manager 130 run in separate processes and could perhaps run on separate physical machines connected over a network. The motion event providers 132 and motion event manager 130 thus use the system for the inter-process communication provided by the operating system to transmit the event tokens from the event providers 132 to the motion event manager 130.

The motion event manager 130 notifies the motion control component 150 when the event token is received and the identity of the event token. The action taken by the motion control component 150 upon receipt of an event token depends upon the nature of the event token. The received event token may contain or identify a particular control command, and the motion control component 150 can simply pass that control command to the target device 122. The received event token may contain or identify a particular media command, in which case the motion control component 150 may be required to convert the media command into a control command capable of being run by the target device 122. Another event token may start, stop, or otherwise control a separate application program run by the motion control component 150.

In the exemplary system 120, the association of motion media with event tokens is preferably made by the motion event manager 130. This association is typically represented by a table, spreadsheet, or other data storage means capable of defining relationships between event tokens and motion media. Upon receipt of each event token, the motion event manager 130 will identify the motion media previously associated with the received token and send the identified motion media to the motion control component 150 for control of the target device 122.

With the foregoing understanding of the basic operation of the system 120, the details of this exemplary motion control system 120 will now be described.

The motion event manager 130 handles the creation of each event provider 132 installed on the system by creating in-proc providers or injecting other providers into their target processes. The event manager 130 also catches events fired from each provider 132 and initiates the appropriate motion request for each event. In the exemplary system 120, the event manager 130 is the only object that communicates directly with the motion control component 150, as will be described in further detail below. The exemplary event manager 130 is accessible by double clicking its icon in the Windows System Tray in a conventional manner.

The purpose of the event handling component 152 is to handle the inter-process communications between the motion event manager 130 and the motion event providers 132. The exemplary event handling component 152 is or may be a conventional software object referred to as a message pump.

The motion event provider objects 132 are individually designed to monitor user configurable events from a given source. The exemplary system 120 employs two types of motion event providers 132: simple in-proc servers 132a,b hosted by the motion event manager 130 and specialty DLLs 132c,d that are injected into a target process to monitor event cases. Each motion event provider object 132 also contains an event configuration control 154 that, as will be described below, allows a user to configure all events supported by the motion event provider objects 132. The motion event provider objects 132 notify the motion event manager 130 of each event caught by the objects 132.

The motion event manager 130 and motion control component 150 operate together to allow interaction between the motion event providers 132 and the target device 122.

The motion control component 150 may be or incorporate parts of a software system as disclosed, for example, in U.S. Patent Nos. 5,691,897 and 5,867,385. The systems disclosed in the '897 and '385 patents are capable of generating device-specific control commands based on hardware independent media commands written to a predetermined application programming interface.

As an alternative, the motion control component 150 may act as a conduit that passes device-specific control commands and query responses between the motion event providers 132 and the target device 122. A motion control component implemented in this manner would not convert between hardware independent media commands and device specific control commands.

A preferred implementation of the motion control component 152 would be to be for the component 152 to function in both a translation mode and in a pass-through mode. In the translation mode, the component 152 converts media commands into control commands. In the pass-through mode, the component 152 simply passes control commands from the motion event providers 132 to the target devices 122. In either mode, query responses are returned from the target devices 122 to the event provider 132 in an appropriate format.

The motion event configuration interface 134 is preferably a visual interface displayed on a screen to allow a user to configure all motion event providers 132 installed on the system 120. The exemplary interface 134 also provides access to the motion web site 136 where new motion media and motion event providers 132 may be downloaded and installed.

As will be described in more detail below, the configuration options allowed by the interface 134 include the ability to enable/disable event providers 132 and map motion media to particular events supported by each provider 132. The interface 134 also provides access to the motion web site 136, allowing for new motion media and motion event providers 132 to be downloaded and installed onto the current system.

Each motion event provider 132 contains a visual configuration control 158 that allows the user to configure the events supported by each provider 132. The exemplary configuration controls 158 use the media view control object 156 to gain access to the available motion media in the motion media file 144 that can be mapped to each available event.

These controls may also be configured to allow the user to add new, customized events to the motion event providers 132. The dynamic events can be defined using parameters such as text (usernames, messages, email, etc.), date/time, or any other parameter particular to an event provider's event source.

The media view control object 156 provides access to all installed motion media scripts as represented by the motion media file 144. Preferably, the media view control object 156 displays a conventional browse/select dialog to allow identification and selection of the available motion media. This object 156 is used by the event provider configuration controls 158 and allows the configuration controls 158 to remain independent of the motion media format.

The media creation tool application 138 allows the user to customize and/or create motion media. This application 138 preferably implements a graphical, easier to use, front-end user interface design.

The motion web site 136 provides a location for the user to download new motion media as well as new and/or updated motion event providers 132. The motion media is preferably stored in a single meta file. New motion media downloaded from the motion web site 136 will be added to this meta file.

The present invention is preferably embodied using the Windows registry; typically, a component category is created for each of the motion event providers 132, allowing the motion event manager 130 to enumerate all providers 132 installed on the system. Primary event sources 132 are user actions (in any active application supported via a motion event provider) and operating system tasks.

With the foregoing understanding of the modules that form the exemplary system 120, various scenarios in which these modules typically interact will now be described.

Referring now to FIG. 2, depicted therein is the scenario describing the startup process of the motion event manager 130 of the system 120. Each of the steps of this startup process will now be described with reference to FIG. 2.

The motion event manager 130 process 130 begins on system startup. The motion event manager 130 process 130 next queries the MOTION_EVENT_PROVIDER component category in the Windows

Registry to enumerate all motion event providers 132 installed on the system.

Third, the registry entry of each of the event providers 132 contains startup information indicating if the particular event provider is either a standard in-proc provider or a specialty provider that is injected into a target process to monitor event conditions.

Fourth, the motion event manger 130 creates a new instance of each provider 132. If the event provider 132 is a specialty provider that is injected into a target application process, the event manger 130 will read the target-process information from the provider's registry entry, find the target process, and perform the DLL-injection. If the target process is not active, the motion event manager 130 will continually monitor the creation of new applications, and perform the injection when/if the requested application is launched.

Fifth, once the event providers 132 are created, the motion event manager 130 will send the appropriate initialization information to each provider 132, including callback information to allow the event providers 132 to post event messages back to the event manager 130.

Finally, the event provider 132 reads initialize message data and establish the necessary event syncs to monitor the events. The initialize message data includes a registry key identifying the location of the event configurations and preferences as last set by the motion event configuration interface 134 or the default installed set.

Referring now to FIG. 3, depicted therein is the DLL injection scenario map. This scenario describes the process of injecting a motion event provider DLL into the address space of a target application.

As shown in FIG. 3, the first step of this process is for the motion event manager 130 to determine which process into which the motion event provider 132 must be injected based on the registry entry of the provider 132.

Once the target process has been identified, the next step is for the event manager 130 to install a Windows message hook in the target process. This causes the event provider DLL to be loaded into the target address space of the target process. The event provider DLL has now been loaded into the required process, and will now wait for the "initialize" message from the motion event provider 132.

Referring now to FIG. 4, depicted therein is the motion event configuration scenario map. This scenario map describes the process of configuring motion events of each of the motion event providers 132.

First, the user launches the motion event configuration interface 134 from system tray interface of the motion event manager 130.

Each event provider object 132 supports a custom visual control 154 that can be used to edit the object's supported events. The event configuration interface 134 creates and hosts these visual controls 154 for the user.

Next, when the event provider configuration control 154 is created and initialized, it will receive the location in the Windows Registry 142 of its persisted event data, which will be loaded into the control 154.

Next, the user will select an event provider 132 to configure. Individual events may be mapped to motion actions, and particular events (as well as the entire event provider itself) may be deactivated if desired. As noted above, these event configuration controls 154 may also provide the means to add additional, customized events based on input user parameters (custom text strings, buddy chat names, email messages, etc).

When the user selects an event to configure, the event provider configuration control 154 will defer to the media view control object 156. The media view control object 156 displays all available motion media via a dialog box, allowing the user to make a selection.

Finally, once the user makes a motion media selection, the media view control object 156 returns data back to the event provider configuration control object 154 (including human-readable description text of the event for display as well as a data token which can later be used to identify media selection). The configuration control object 154 then persists this information to the Windows Registry 142.

Referring now to FIG. 5, depicted therein is a motion event scenario map. This scenario describes the process of handling a motion event.

The scenario depicted in FIG. 5 begins whenever an event occurs. The occurrence of an event may be caused from a user action, operating system event, or an event situation monitored in a third-party application.

The event provider 132 then fires a event token associated with this event to the event manager 130. The event token has previously been stored in the registry during the event configuration process. If the provider 132 requires queried data to be returned, the provider 132 will also pass the necessary callback data to the event manager 130.

The event manager 130 next receives the event and passes the requested media information to the motion control component 150. The motion control component 150 then executes the specified motion media on the target motion device 122.

Finally, if a query action was requested, the motion control component 150 will return the appropriate data. The motion event manger 130 will send the data through the specified event provider callback mechanism.

Depicted in FIG. 6 is a motion web site scenario map. The scenario of FIG. 6 describes how a user may obtain new motion media and motion event providers 132 from the motion web site 136.

This process may be started when users visit the motion web site 136 to browse currently available motion event providers 132 and new motion media. In FIG. 6, the reference character 144a is used to represent a motion media file stored locally by the system 120, while the reference character 144 represents a motion media file stored at a remote location.

Next, the user selects the desired provider/media download option, and the new software is installed into the motion event manager 130 including the motion control component 150.

The next time the motion event configuration interface 134 is launched, the user will be able to configure the new event provider 132 or motion media in the local motion media file 144a.

Alternatively, users may download new event providers 132 and motion media directly from within the motion event Configuration dialog interface. This configuration dialog will provide the following options to the user: Download new Motion Media and/or Download/install new motion event providers 132. The user or the motion event manager 130 may also check for new versions of currently installed motion media and/or event providers 132.

Next, the user selects the desired provide/media download or update option, and the configuration dialog interface object 134 automatically downloads and installs the new software from the media web site 136.

Finally, once the new software is installed the configuration dialog 134 will automatically update to provide access to the new components and/or media.

Referring now to FIG. 7, depicted therein is the system shutdown scenario map. This scenario describes the process of shutting down the exemplary event manager module 130 associated with the motion control component 150.

Upon operating system shutdown, the motion event manager 130 will prepare to terminate.

The event manager 130 next posts a shutdown message to each event provider 132 currently being managed by the manager 130. Specialty event providers 132 that have been injected into external processes may have already finished if the target process has been shutdown. In this case those event providers 132 would have already notified the event manager 130 that they are no longer available.

Each event provider 132 performs any required shutdown tasks.

Upon finishing any shutdown tasks, each provider 132 will notify the event manager 132 that the provider 132 is now finished.

Once the event manager 130 receives notifications that each of the event providers 132 managed thereby have been shutdown, the event manager 130 itself is now finished.

Referring now to FIG. 8, depicted therein is a scenario map illustrating the mapping of text to motion events. This scenario generally describes the mapping of text based event situations to motion.

The first step of this process is for a text based event situation to occur. This text could be one or more of the following: (a) a particular sub-string in an Instant Message or the entire message string itself; (b) an Instant Message sent from a target screen or 'buddy' name; (c) a text string converted from a speech-to-text engine installed on the user's machine; and/or (d) an email message meeting previously configured criteria (Sent From, Subject, message content, etc). In the case of an event created using a peer-to-peer networked application such as Instant Messenger-type process, text is entered at a remote source application and sent as a text message to a receiving application.

The motion event provider 132 monitoring the specific text based event catches the event, and performs any pre-processing of the text required to identify the particular event. In the peer-to-peer application described above, a DLL functioning as the event provider 132 is injected into the receiving application; the DLL event provider 132 intercepts the message received by the receiving application and treats the received message as an event.

Once an event has been identified, the event provider 132 will lookup the corresponding motion event token as previously configured. As generally described above, the motion event tokens are pre-loaded upon initialization. In the peer-to-peer example described above, the DLL functioning as the event provider 132 sends the text message as part of the event token to the event manager 130 using a process-to-process communication system as generally described above.

After the event token containing the text message is sent to the motion event manager 130, the event manager 130 determines the type of event represented by the received token.

If the event manager 130 determines that the received event token corresponds to a text event, the event manager 130 next parses the text parameter from the event token. The motion event manager 130 looks up the motion media associated with the event type and event text parameter. The appropriate motion media is then sent to the motion control component 150 for execution on the target motion device 122 as described above with reference to FIG. 5.

The process described above with reference to FIG. 8 can also occur in the reverse. In particular, the event manager 130 uses the motion control component 150 to continually query the target device 122 for state information. When the state information meets certain parameters, the control component 150 causes the event manager to 'create' a new event (such as a text event) and send it to an event provider 132.

The event provider 132 in turn then causes the receiving application to sent a message to a remote peer-to-peer application; again, the receiving and remote peer-to-peer applications may be Instant Messenger compatible applications.

An example of the use of the present system to verify motion status would be for the event manager 130 to continually or periodically query the target device 122 through the motion control component 150 for status parameters that indicate a fault condition. Once a fault condition occurs, the event manager 130 builds a text message that describes the fault and then sends the text message to a remote application, such as an Instant

Messenger user, using process-to-process communication system and the peer-to-peer networked application.

The following Table A describes the interface specifications for components of the exemplary motion control system 120 described above using the exemplary motion control component 150.

TABLE A

component	interface	method(s)/description
motion event manager 30		
	IXMCEventProviderMgt	
		EnumerateProviders used by configuration dialog object to get a list of all installed motion event providers 132
		EnableProvider() used to enable/disable given event providers. will be called from the configuration dialog object
	IXMCEventScheme	
		GetSchemes() used by configuration dialog object to get current list of schemes on the system
		AddScheme() – used by configuration dialog object to add a new scheme
		RemoveScheme() – used by configuration dialog object to remove a given scheme

		RenameScheme() - used by configuration dialog object to rename a given scheme
Event configuratio n controls 158		
	IXMCEventConfig	
		Initialize() – called by the configuration dialog object which hosts this control, used to pass required init data, such as 1) current registry location where event configuration data may be loaded/persisted, 2) interface to the media view control 156, etc..
Media View Control 156		
	IXMCMediaViewCtrl	
		method: SelectMedia() – called by each event configuration control. This method will display a dialog to visualize all motion media available on the system and return a tokenized data param that can later identify the media item selected

In the following discussion, the details of a number of the exemplary components of the system 120 will now be described.

The Windows Registry 142 is used to persist event management with motion control configuration data.

Data associated with the motion event providers 132 is persisted to the Windows Registry 142 primarily to accommodate event provider DLLs that need to be injected into a target process, an environment where those DLLs may not be able to rely on standard COM storage alternatives.

At runtime, when the motion event manager 130 initializes an event provider, the provider will receive a location in the registry where it should read its previously configured event data.

At design time, when hosted within the motion event configuration interface 134, the event configuration controls 156 of the event providers 132 receive the registry location, and will persist configuration changes to that location.

Event providers will persist both standard (hard coded) and custom events to their registry storage location. Associated with each event will be the configured motion event token data, which the provider will use when firing event notifications back to the motion event manager 130.

The motion event manager 130 manages all registry locations for the event provider objects.

The event manager provides a mechanism to support different event schemes. This allows configurations unique for several different users, situations, or motion devices. When a new scheme is selected via the motion event configuration interface 134, the event manager 130 will pass the new scheme registry location to each event provider object 132, allowing access to the new scheme data. Each scheme will be located in a unique sub key under the primary event manager registry location.

The user interface components 154 and 156 of the system 120 may be implemented in many different forms. The motion event configuration control 154 is used to configure all motion event providers 132 installed on the system, as well as to provide access to the motion web site 136 where new motion media and motion providers may be downloaded and installed. The interface of the media creation tool 138 is a preferably graphical representation of the motion device or system, where simple drag-drop, click, and record operations will facilitate the creating and modification of motion media.

The system 120 is designed to be easily extendible via new motion event provider and motion media components. The system 120 is also capable of supporting any number of additional event sources with the addition of new motion event providers 132 by registering these new providers with the system 120. These event providers 132 can link to virtually any event source to a target system 122.

The motion media formats can be modified and extended without requiring changes to any event provider objects. For example, a XMCMediaCtrl object proxies the raw motion media format to the event providers. Accordingly, once the XMCMediaCtrl component is updated to handle any new media formats, the event providers 132 may, by design, make use of the changes.

Instead of triggering entire motion programs as described above with reference to FIG. 6, a motion program or media set may be streamed to the device 122 through the system 120. Whether or not and how the media plays may be controlled by captured events. For example, a media player may fire events based on different musical notes or tones. When such events are received, one media stream may be selected over another, thus causing the motion device 122 to perform different actions. This may also occur with the standard motion programs described above.

Very large motion programs may be downloaded in partial form and then either downloaded in full over time or as certain specified events occur in the system.

Similar to streaming media support, single motion operations may also be supported in which a specified event causes a single motion operation to take place immediately. One example of this would be an event that causes movement of one axis in the clockwise direction.

The present invention may be embodied in forms other than those described above. The scope of the present invention should thus be determined with reference to the following claims and not the foregoing exemplary detailed description.